Dustin Horvath
2729265
EECS 665 Lab 06

The purpose of this lab is to pair the Lex scanning tool with the Yacc parsing tool to output a ‘dot’ file containing code to produce a digraph of all the function calls made within a given program. The dot language is fairly simple, and any nuances or cadences of the language are not strictly needed to produce the basic output needed for the programs being supplied. For this lab, simply emulating the output exampled in the lab assignment is sufficient to produce working code.

The scanning program Lex is fairly straightforward to implement. Lex operates by being supplied particular scanning rules that define the tokens it produces. Each rule in the ‘.l’ file is compared against scanned input. The input string is then *tokenized* and a token table is produced containing the actual string values associated with each token. Rules are prioritized and evaluated in a top-down strategem, so rules evaluated at the top of the list have higher priority than rules lower in the rule list. Scanning is an important step of compiling because it converts an arbitrary string into a discrete list of known token types. Input strings that contain illegal characters can either be discarded, or errors can be produced.

The parsing program Yacc is more complex, but uses a similar rule format to Lexx. It’s rules are also evaluated top-down in precedence, and it requires a supplied set of rules for each parsing format. Yacc takes a list of tokens as input, and attempts to parse those rules into an abstract syntax tree. Yacc can only parse LALR-type languages, so rules cannot be left-recursive; Yacc will either yell at you with errors, or discard some of your rule set with a warning. Given a particular list of tokens, Yacc uses its rules to determine whether or not a particular set of tokens is part of the language those rules represent. It scans through the tokens (Lex scans through input *symbols*) and can produce any output as determined by the rules.